DOCUMENT RESUME

ED 446 903 SE 063 721

AUTHOR Jackson, Shari L.; Stratford, Steven J.; Krajcik, Joseph S.; Soloway, Elliot

TITLE Model-It: A Case Study of Learner-Centered Software Design for Supporting Model Building.

SPONS AGENCY National Science Foundation, Arlington, VA.; Michigan Univ., Ann Arbor.

PUB DATE 1996-00-00

NOTE 13p.

CONTRACT RED-9353481

PUB TYPE Reports - Research (143)

EDRS PRICE MF01/PC01 Plus Postage.

DESCRIPTORS *Computer Simulation; Computer Software Development; *Computer Uses in Education; Educational Technology; High Schools; Instructional Design; *Instructional Materials; *Models; Multimedia Materials; Science Education; *Scientific Methodology

ABSTRACT

Learner-centered software design (LCSD) guides the design of tasks, tools, and interfaces in order to support the unique needs of learners: growth, diversity and motivation. This paper presents a framework for LCSD and describes a case study of its application to the ScienceWare Model-It, a learner-centered tool to support scientific modeling and simulation. The complex task of constructing and testing models helps scientists develop their understanding of natural systems. With Model-It, students use learner-centered modeling tools to easily construct, verify, and analyze qualitative models, using a dynamic and photo-realistic interface. Findings suggest that high school students who know nothing about computer modeling can quickly build and test simple models, and that various components of the software design intentionally support the task of modeling by meeting specific learner needs. (Contains 11 references.) (Author/YDS)

Reproductions supplied by EDRS are the best that can be made
from the original document.

# Model-It: A Case Study of Learner-Centered Software Design for Supporting Model Building

*Shari L. Jackson, Steven J. Stratford, Joseph S. Krajcik, Elliot Soloway*

University of Michigan

1101 Beal Ave.

Ann Arbor, MI 48109, USA

E-mail: {sjackson, sstrat, krajcik, soloway}@umich.edu

ED 446 903

## ABSTRACT

Learner-centered software design (LCSD) guides the design of tasks, tools, and interfaces in order to support the unique needs of learners: growth, diversity, and motivation. This paper presents a framework for LCSD and describes a case study of its application to the ScienceWare Model-It, a learner-centered tool to support scientific modeling and simulation. The complex task of constructing and testing models helps scientists develop their understanding of natural systems. With Model-It, students use learner-centered modeling tools to easily construct, verify, and analyze qualitative models, using a dynamic and photo-realistic interface. We show that high school students who know nothing about computer modeling can quickly build and test simple models, and that various components of the software design intentionally support the task of modeling by meeting specific learner needs.

**KEYWORDS:** Educational applications, Science applications, Modeling, Simulation, Multimedia, Learner-Centered Software Design

## INTRODUCTION

Modeling software designed according to *user-centered* principles helps scientists to construct and test models with speed, efficiency and accuracy. The expert scientist recognizes the scientific benefits of modeling, and chooses usable software to assist his/her investigation. However, the same software may not be suitable for the student, who may be unfamiliar with modeling or computer-based models. Learners needs encouragement, individualized instruction, and someone to shoulder some of the intellectual burden; learners especially need software designed to meet these unique needs. *Learner-centered* software design is the new challenge for human-computer interaction, with the goal of providing support for both *learning* a task and *doing* it [9].

In this paper, we present a framework for learner-centered software design (LCSD) and describe its critical considerations. We present the ScienceWare Model-It as an example of the application of LCSD principles. Model-It has been designed to provide a supportive environment in which students can learn to construct simple process flow models. In our preliminary classroom testing, high school students easily used Model-It to build and explore models in our initial target domain of stream ecosystems. We show how Model-It's design impacted this outcome.

## Learner-Centered Software Design

Software design requires addressing three top-level issues:

- Tasks: What are the tasks the software will support?
- Tools: What tools will perform these tasks?
- Interfaces: What is the interface to those tools?

PERMISSION TO REPRODUCE AND DISSEMINATE THIS MATERIAL HAS BEEN GRANTED BY
S. Stratford
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)



U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☒ This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

SE 063 721



BEST COPY AVAILABLE

User-centered system design (UCSD) [5] addresses these issues by focusing on the needs of all users. In contrast, learner-centered software design considers the user to first be a learner with unique needs. Table 1 is a comparison of differing software needs of experts and learners from the point of view of each design methodology:

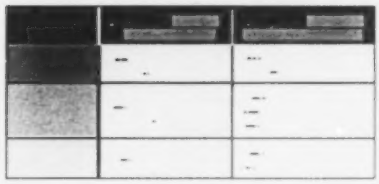

Table 1: Differing software needs: UCSD vs. LCSD

For experts to carry out their tasks, they need useful, productive software designed with usable interfaces. Learners have the same usability needs, but they also require additional support for learning. Following is a discussion of each of the unique needs of learners addressed by LCSD:

*Support for learning while doing:* Learners need support to learn how to do the task for which the software is designed, and to learn from the doing of it. Learning the task is not the same as learning how to use the software; not only do learners need to understand how to use the software, but also how and why to do the tasks for which the software is designed.

*Tools appropriate for their style of learning and level of expertise:* We recognize that different learners may have different styles of learning and require different tools to accomplish similar learning tasks. For example, some learners may prefer to work verbally, while others prefer pictures, diagrams, or equations. Other learners may come to the task with higher levels of expertise, and therefore may prefer a tool with different capabilities or modes of operation than needed by the beginner.

*Engaging interface that structures the task:* The learners' first contact with the software is with the interface. It must be engaging or they may not be motivated to learn the tools and tasks. The interface should also visibly structure and organize the tools and the task to guide learners in figuring out what to do, and furthermore, to encourage them to think about what they are learning.

## Software as Scaffolding

We implement LCSD by providing "software-realized scaffolding" [2]. "Scaffolding" is an educational term which refers to providing support to learners while they engage in activities that are normally out of reach [6, 11]. Software-realized scaffolding of tasks, tools, and interfaces provides support as learners gain in expertise and understanding (Table 2):

Table 2. Software-realized scaffolding of tasks, tools, and interfaces

• To scaffold the *tasks* , constrain the complexity by providing a simpler set of tasks for the learner to perform. Defaults and limited options can initially simplify the task, to be expanded when the learner is ready.

• To scaffold the *tools* , provide a range of tools which support different styles of learning and different levels of expertise.

• To scaffold the *interface* , make it engaging: it should be novel, should pique curiosity and interest, and should be aesthetically pleasing. The interface can also be scaffolded by providing a visual structure for using the tools and performing the tasks, and by encouraging reflection by providing opportunities for learners to articulate what they are learning while they are learning it [3, 8].

The ScienceWare Model-It was designed using LCSD principles. In the next section, we describe the value of

modeling as an educational activity. Next, we present the design of Model-It and how we incorporated software-realized scaffolding.

## Modeling: A Constructive Learning Activity

Scientists build models to test theories and to develop a better understanding of complex systems [4]. We believe that students can similarly benefit from building models. This approach is consistent with current conceptions of learning in which students actively construct understanding by creating external representations of their knowledge [7].

Model-It was therefore designed to support the two main tasks associated with scientific modeling: model construction and model verification. Model construction refers to the implementation of the model, defining the relevant objects and specifying the relationships between them. Model verification refers to testing the model, running simulations to test that it actually works as intended. Moreover, we have the higher-level goal of supporting students learning about modeling: acquiring strategies for constructing and verifying models, and developing the skills to plan, predict, and debug them.

The software modeling tools that are currently available for students fall into two major categories: pre-defined simulations, and modeling environments.

*Pre-defined simulations:* (e.g., Maxis' SimCity, Wings for Learning's Explorer) While these programs are user-friendly and provide a great deal of depth in their pre-programmed domains, they do not provide access to the underlying model that drives the simulation, nor the ability to add or change functionality. Students learn about the target domain by using these programs, but they do not gain the benefits of actually constructing the model themselves.

*Model ing environments:* (e.g., High Performance System's Stella, Knowledge Revolution's Working Model) These programs, designed for the task of modeling, allow great flexibility in building models. However, they are difficult to learn; building complex mod els requires a substantial commitment of time and effort to learn a complex authoring language [10]. There is insufficient scaffolding of the task for the learners we wish to support; the tools are not adaptable to middle and high school students with little mathematical background.

Model-It has been designed to combine the engaging aspects of a SimCity-like environment with some of the functional capabilities of Stella. Model-It is intended to be an entry point for learners into modeling activity; one potential outcome is that as students gain sufficient expertise and understanding of model building, they may transition into more sophisticated environments such as Stella.

## MODEL-IT

We designed Modei-It to provide learners with the combined flexibility and ease of use necessary in a computer-based learning tool. In particular, Model-It design incorporates scaffolding to address learner's needs regarding software tasks, tools, and interfaces, as summarized in Table 3, below.

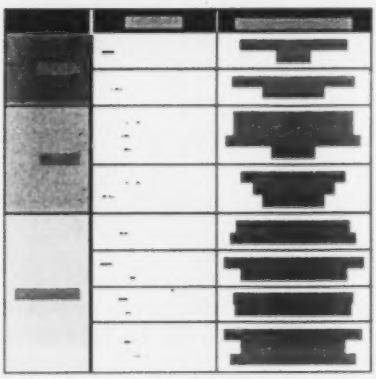

Table 3: Learner-centered features of Model-It

Model-It can be used to build a wide range of process flow models; for our preliminary classroom study we chose the domain of stream ecosystems. In our description of the program, we use examples from this domain.

***Tasks:*** With Model-It, we scaffold the task primarily by defining the modeling task as a qualitative one and by hiding the complexity of the tasks involved in building models. To build a model, students select from a set of high-level objects, define the factors (measurable quantities or characteristics associated with each object), and define the relationships between the factors. For example, in our example domain of stream ecosystem modeling, students might start with the stream object, define its factors "phosphate" and "quality," and then define the relationship between them, all in a matter of minutes.

***Tools:*** Learners need tools that adapt to their level of expertise, so Model-It provides a range of ways to define relationships. Initially, relationships can be defined *qualitatively* by selecting descrip tors in a sentence, e.g., "As stream phosphate increases, stream quality decreases by less and less" (Figure 1). As students' skills increase, and their needs grow more sophisticated, they have the option of defining the relationship more *quantitatively* , by entering data points into a table (Figure 2).

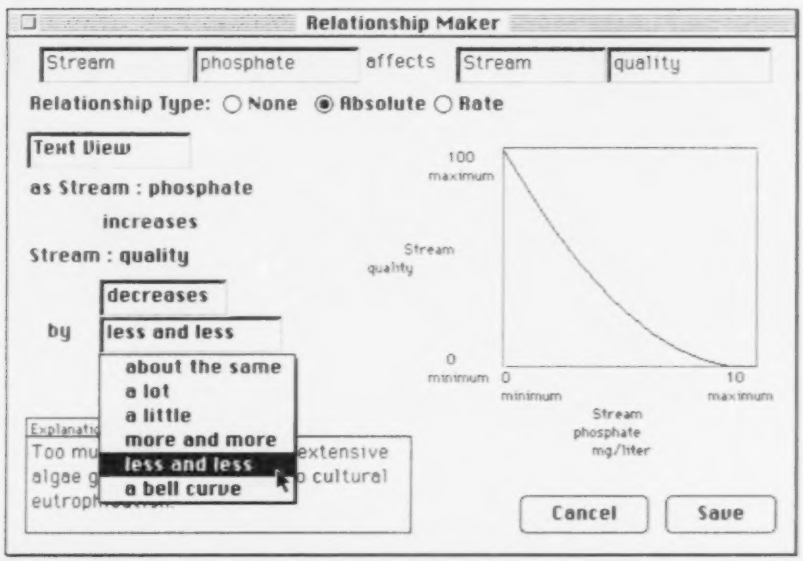


Figure 1: Defining absolute relationships: Text View

BEST COPY AVAILABLE

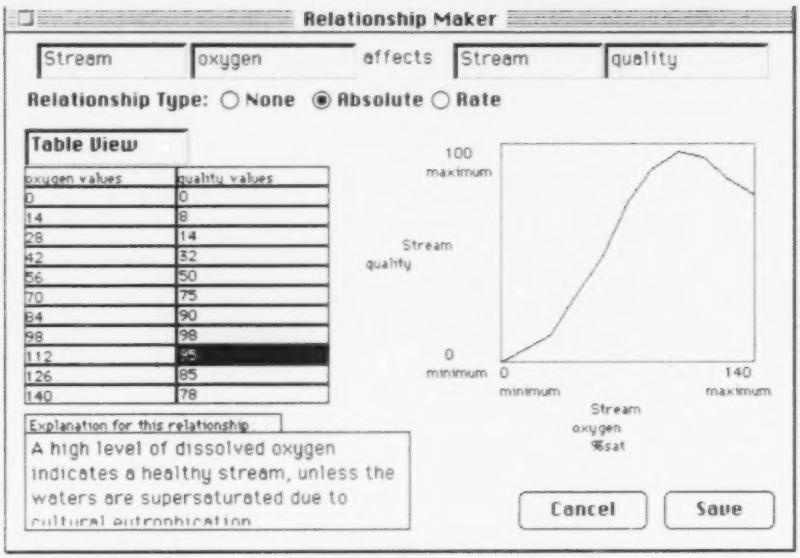


Figure 2: Defining absolute relationships: Table View

To support different learning styles, these tools provide a variety of ways to visualize relationships. For example, given a qualitative definition, the software translates the text into a quantitative representation; e.g. "decreases by less and less" is interpreted as shown by the graph in Figure 1. This visualization of the text reinforces students' understanding of how to "read" a graph.

The above relationships are termed "absolute," in that the value of the affected factor is completely ("absolutely") determined by the value of the causal factor. Model-It also supports the definition of rate relationships which define feedback equations representing the rate of change of a factor. We similarly scaffold the definition of rate relationships by providing a qualitative representation: e.g., "At each time step, and for each mayfly, add mayfly rate of growth to mayfly count."

***Interfaces:*** Learners often need extra motivation to sustain interest in a task, and the interactivity and engaging personal graphics of Model-It can help provide that motivation. Students run simulations to try out experiments using their models, such as exploring the impact of increased phosphate levels on over all stream quality. The objects in a model are displayed using photo-realistic graphics, and during a simulation, graphical meters provide real-time feedback of changing values (Figure 3). For our classroom study, the background graphic is a photograph of the actual stream the students stud ied. By using a photo of *their* stream we expect to make the task more concrete and authentic for the students; such meaningful, personal tasks are more motivating for students [1].



BEST COPY AVAILABLE

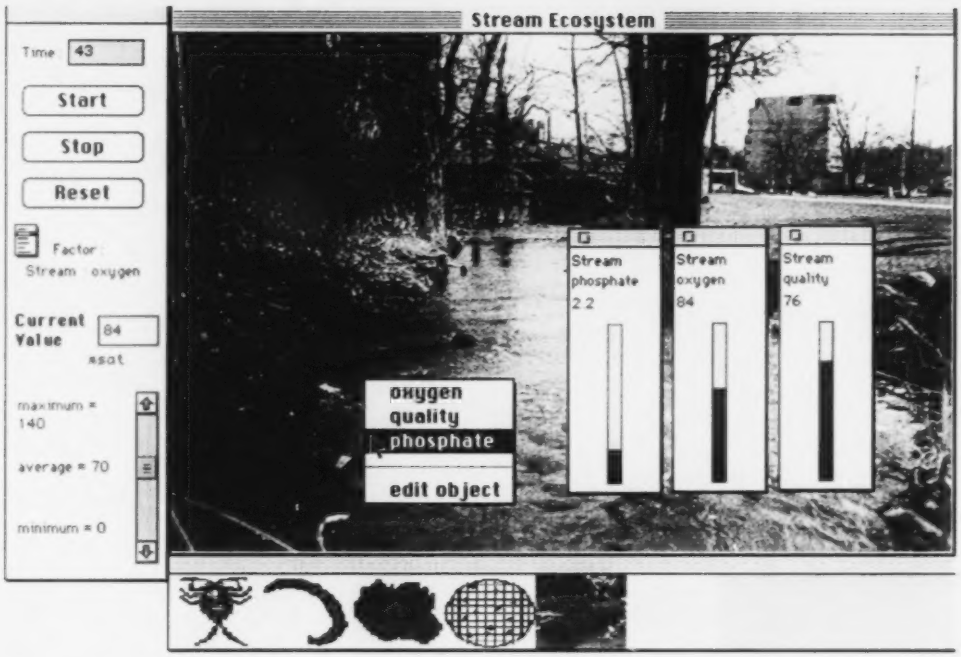


Figure 3: Running simulations

The interface also provides structure for the task, and encourages reflection about the task. To structure the task and guide the learner, we offer specific and useful options (e.g. the pull-down menus in Figure 1, tabular data entry in Figure 2), as opposed to unrestricted text or equation entry. To encourage reflection, Model-It elicits articulation from the students by providing an "explanation" field (e.g. Figure 2) where students can type in an explanation for each relationship they create.

# RESEARCH QUESTIONS

For our preliminary classroom testing, we first considered Model-It's fundamental goals: supporting model construction and verification, and supporting the learning of modeling strategies and skills. To measure achievement of these goals, we defined two criteria:

• *Quality of Building and Testing Models:* Could students build good quality models in a short period of time? What strategies did students develop for testing their models? Was there a relationship between the quality of their testing and the quality of the resultant model?

• *Conversations as Evidence of Understanding Modeling:* How did students think and talk about modeling while engaged in modeling activities? How do student conversations reflect their understanding of the modeling process?

Furthermore, to evaluate the success of the specific software features designed to provide scaffolding for tasks, tools, and interfaces (Table 3), we ask the question: What is the impact of each of these features on the behaviors of the students?

# Methods

Our testing took place within the overall context of the ScienceWare project: a research project in collaboration with science teachers to develop, pilot, and assess a three year high school science cur riculum emphasizing modeling, project-based activities [1], and routine use of computing tech nology. The pilot class contained 22 ninth grade students who were given everyday access to Macintosh color powerbooks, commercial and research software, and various multimedia equipment. The students spent several months investigating ecosystems;

specifically, the ecosys tem of a stream that runs behind their school. They collected a variety of biological, physical, and chemical data to determine the quality of the water. At the end of the school year, near the end of their project, they used Model-It to build computer-based models of the ecosystem they had studied.

Students worked in groups of two with Model-It, over a period of one week. For three of the days, they used a printed guide (along with the program) designed to introduce them to modeling, teach them how to use the program, and help them design and test some simple models. Students were encouraged to predict, evalu ate, and elaborate as they learned to construct their models.

On the fourth day, they were assigned an open-ended modeling task to create their own models to represent one of three choices of ecological phenomena:

• Biodiversity: the impact of pollution on some stream macroinvertebrates (mayflies and midge flies);

• Cultural eutrophication: an excess of phosphates leading to algal blooms;

• Land uses: the impact of man-made structures such a golf course or parking lot on stream quality.

Teachers and researchers were available to give advice and to answer questions. On the fifth day, a researcher led the class in a discussion about the models they constructed, giving them an opportunity to describe and discuss their models with the class.

# Data Analysis Techniques

Several data sources were used: the models the students constructed; soft ware-generated log files of each group's activities with the program; and audio and videotapes of two groups' conversations and computer activity. We used several different methods for reducing and analyzing the data: concept maps of students' models, patterns in log files, and analysis of conversation transcripts.

## Concept Maps of Students' Models

On the fourth day, students created their own models. To assess the quality of these models, we drew concept maps corresponding to the computer models each student group had built. Figure 4 shows an example of a concept map we drew for one group's model.

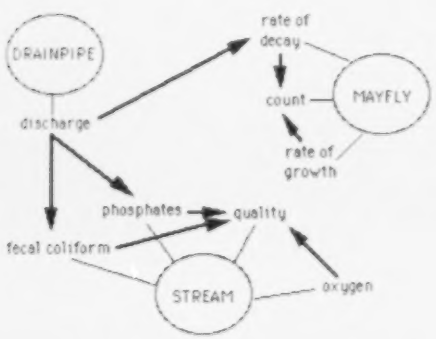


Figure 4: An example concept map of a student model

Objects are represented with circles, and their associated factors are named and attached to those circles with lines. Relationships between the factors are represented as lines with arrows indicating the direction of the relationship. For example, looking at the upper left corner of Figure 4, we see that this group created a factor "discharge" for the drainpipe object, and defined relationships from the drainpipe discharge to affect the stream's fecal coliform and phosphates, and the mayfly population's rate of decay. For simplicity, the specifics of each relationship are not included in Figure 4, but were noted for our research; for example, the relationship between drainpipe discharge

and mayfly rate of decay was defined by the students to be: "As the drainpipe discharge increases, the mayfly rate of decay increases by more and more."

We evaluated the reconstructed models and compared them with appropriate scientific models, judging qualitatively for accuracy and completeness. We used a scale from poor to excellent, on a scale of 1 to 4: [excellent (4) accurately represented the modeling task, without errors or misconceptions; good (3) mostly accurate, but incomplete; fair (2) some reasonable relationships, but incomplete or in error; poor (1) no accurate representations]. The model in Figure 4 was judged a good model, because the relationships the group created were reasonable and demonstrated that group's chosen task of showing that mayflies were pollution-sensitive; however, they neglected to represent the impact on midge flies, as the scenario specified.

### Patterns in Log Files

To assess the quality of students' testing strategies, we examined the log file data to see the order in which students built and tested their factors and relationships, and to evaluate how well they conducted testing which would exercise all of the relationships they created, over the range of possible values of the causal factors. We used a scale from none to excellent, on a scale of 0 to 4: [excellent (4) tested all relationships over the range of possible values; good (3) tested all, but incompletely; fair (2) missed testing some; poor (1) very little testing; none (0) no testing].

Using the time stamp information in the log files, we also reviewed this data to determine the amount of time students took to define a factor or relationship, or run a simulation.

### Analysis of Conversation Transcripts

To see students' conversations about modeling, and to evaluate the success of the scaffolding features of Model-It (from Table 3), we looked in detail at conversation transcripts of two pairs of students throughout four days of testing, as well as the classroom conversation on the fifth day. We combined the conversation transcripts with log file data in order to match student dia log with actions they took on the computer. Conversational episodes which reflected the impact of particular Model-It features were identified.

## Results

### Quality of Models and Testing

Two-thirds of the groups in the class created reasonably good quality models in one class period (quality rating 2.5 or above). The average rated model quality was 2.7.

Testing quality averaged fair to good (2.3). Observing students' testing strategies, we saw that testing activities often suggested and led to the discovery of flaws in their model or the addition of new relationships to the model. For example, one group had erroneously defined new factors (bacteria and algae) which were already pre-defined as objects. When they tested their model, they observed behavior which didn't agree with how they thought it should work, and consequently discovered their error. Another group, while testing their model, realized that it would make sense to link two more factors together by creating another relationship, so they went back and defined the new relationship.

Good testing strategies therefore appeared to lead to better quality models. There was a moderately strong correlation (.65) between the quality of testing strategies and the quality of the models.

### Conversations as Evidence of Understanding Modeling

For evidence of an understanding of the modeling process, we looked for student conversations involving explanations, arguments, predictions, and evaluations. On the fifth day of testing, the last day of school, students presented their models for class discussion. Students were eager to explain their models, were able to describe their relationships from memory, expressed positive opinions of the software, and gave concrete suggestions for improvements to the program. Several groups argued for different models of the same phenomena, and defended their positions. For example, during a discussion of the relationship between the quality of a stream and the population of pollution-tolerant midge flies, one student argued that an abundance of midge flies would lower the water quality. Another student countered: "The count of midge flies is an indicator of what good water quality is. I don't really think that it brings the water quality down."

For the most part, students who made predictions about the behavior of their models made them in response to directions given in their guide. Of the two pairs we videotaped, one consistently made predictions, and explicitly compared the results of their testing with their predictions, which helped them uncover problems with their model.

In their model building and testing activities, students often reflected on modeling, and how their model reflected real life. As one student said in our preliminary user testing, "it makes you think more about a real-life situation, where's there's no real answer, *you* set it up and everything." Sometimes there was confusion when they expected their model to behave more like real life – even when they hadn't built in the relationships to create that behavior. More often, they appeared to realize that their model was not the real world. For example, while typing an explanation, one girl commented, "This is like us understanding the mechanics of the program, not what's going on in real life." Students also wanted to make their models as realistic as possible. For example, one pair was concerned about setting the size of the golf course to an appropriate number of acres, which led to an extended conversation about estimation of land size.

# Impact of Learner-Centered Design Features

## Task Design: Support for Learning the Task

One of the goals of our software design was to hide the complexity of modeling for students, so that they could concentrate on higher-level concepts in learning how to model. Students were certainly able to create models quickly; one pair created four accurate, interrelated relation ships in four minutes, and in the next four minutes, tested and verified their model, and found another relationship to add.

The idea of "objects in the stream" is central to how the students think about and build their models. These objects are connected with relationships, and students referred to this linking in several object-oriented ways, such as "chains" or "hooks." Students found it easy to place object icons into their model, click on these object icons to access a menu of object-specific commands, and generally to express relationships in terms of things affecting other things.

Redefining the task qualitatively was probably the biggest factor in making relationships easy to create. Students easily expressed their understanding of relationships between objects by choosing qualitative options from the menus to build a sentence (e.g. Figure 1 "as stream phosphate increases..."). Reading over the sentence also appeared to help them confirm that a relationship said what they wanted it to say.

Much of the computation done by Model-It's simulation engine is hidden from students. To make calculations based on the relationships, Model-It converts the qualitative definitions into quantitative functions. Furthermore, students only deal with relationships between two factors at a time; the simulation engine handles the combined impact of multiple relationships on the same factor, by summing up or averaging their effects. Students seemed to find this means of combination natural, and spoke of "balancing" impacts or having "good" and "bad" factors affecting the same factor as a predictor of that factor's behavior.

## Tool Design: Support for Different Styles of Learning

Providing a variety of visualization tools (text and graph views of absolute relationships) proved very useful for learning, as students could use the graph to interpret the text, or vice versa, when deciding how to define a relationship. For example:

Student: Should we say more and more. Cause it gets more... I don't really understand that one. More and more. Should we say more and more? Well, let's look at the graph thing. If the stream temperature is like, really high, then it [oxygen] starts going down.

We also provided both qualitative and quantitative means of defining relationships, to support students at different levels of expertise. While most students exclusively used the qualitative "text view" tool, one student preferred the precision afforded by the quantitative "table view," and used it for all his relationships. He expressed a desire to have even more options for the relationships, for example, a graph with a logarithmic scale.

## Interface Design: Engaging the Learner in the Task

The students spent quite a bit of time with the interactive components of the interface: clicking on buttons, moving

sliders, popping up menus, and moving meters. They took time to investigate the behavior of all of the interactive components. For example, one pair of students took a minute to figure out exactly how the sliders worked. The students spent time lining up the meters, and sometimes put the meters in a visual order on the screen which reflected the underlying model (e.g. left to right, this affects that, that affects that, etc.). One of the first things some of the students did when re-opening a model was display and line up the meters for all of the factors.

Meters provided visualization of simulations as they ran; students often made exclamations such as "whoa, it's going fast!" or "whoa, they both went down," in response to this immediate feedback. Meters were used for model testing and verification. For instance, one student, watching the meters, said "the phos phate went up, algae went up, bacteria didn't go up though, did we define algae and bacteria relation ships?" The interface provided her with information which led to discovering an error in the model, which she later went on to fix.

A familiar picture of the actual stream the students had worked in provided the background graphic for the simulation window. The students recognized and appreciated this picture, and spent time personalizing their model by carefully arranging their object icons on this background. In our preliminary user testing, one student was motivated by the photo to build a model of "how our stream was when we tested it," so she set the factors to the values she remembered from their real-world tests, to see what would happen in her model.

Half of the groups used the explanation field of the relationship window to articulate the relationships they constructed in their final models. Most of their explanations were reasonable and consistent with the relationships they defined, for example: "Phosphorus is a natural plant fertilizer. The amount of algae in the stream greatly increases with the increase in the amount of phosphates in the water." For one pair of students, thinking out the wording of what they were typing in the explanation box seemed to help them understand both what the other was thinking and whether the underlying model accurately reflected what they wanted it to do.

In summary, this research is only the first step towards the development of a learner-centered tool for model building. The preliminary data is encouraging, and we look forward to improving the design based on the results of the user testing. The question that needs to be further investigated is how Model-It supports the student in learning about the domain of stream ecosystems, and developing expertise in modeling and simulation. Over the coming year, we will be working closely with several classrooms of ninth and tenth grade students to observe the impact of long term use of Model-It. As students gain experience over successive investigations, we will be looking to see its influence on students developing a deeper understanding of natural phenomena, and the disposition to see scientific phenomena as analyzable systems.

## Future Directions for Research

We are currently implementing some features which had been designed into the program, but were not implemented in time for the preliminary classroom testing. For example, to support the creation of system models in other domains, and to make the interface more personally meaningful, we will support students defining their own objects, and importing their own graphics and digitized photographs. To provide more visualization support, we are implementing a graphical view of factor values as they change over time during a simulation. And to provide more structure for the task, the program will generate a graphical overview similar to the one in Figure 4.

Other future changes to the program were suggested by the classroom testing. For example, the data showed that frequent and iterative testing of models as they were built tended to result in better models. We should therefore design the interface to guide the learner towards these testing strategies, perhaps with scaffolding to prompt them to test each relationship as they create it. We also noticed that predicting model behavior was useful for students, yet they often didn't make predictions, so we intend to design and implement a tool for scaffolding students' predictions about their model. This tool would support both making predictions before testing a model, and analyzing and explaining those predictions after testing. Other future goals include: eliciting more articulation from students, providing built-in checks for common student errors, and supporting the growth of expertise by providing another level of modeling where other and more complex types of relationships might be defined.

Finally, there are more long-term goals for research with the software, in which we consider other tasks that the program might support. For example, we are looking into networking the software so that students can interact and share their models. Students might even work on different sections of the same model, so that their changes affect each other's model. For example, if the students "upstream" add pollution to the stream, the students "downstream" will see their organisms dying off. We would also like to integrate Model-It with a database of real-world data, so that students can validate their models by comparing simulation results with data from the real world.

# Concluding Remarks

Our research has shown that students can use the learner-centered Model-It program not only to create models of phenomena they have observed, but to create them with relative ease and speed. We found that many of our students were able to create and test somewhat complex qualitative models in one 50-minute class period. We observed that the task of building models with Model-It software engaged most students at a level which enabled them to think deeply about the process of constructing and testing models. While they were constructing their models, they engaged in various high-level cognitive activities such as making explanations, stating arguments, making predictions and evaluations. Model building in particular, and these kinds of activities in general, are activities that are usually inaccessible to learners in high school science classrooms.

On a more abstract level, with Model-It data we have demonstrated some applications of learner-centered design principles to interface design. On a fundamental level, students learn through the act of constructing artifacts which represent their understanding, and reflecting on those artifacts. By providing a flexible tool for constructing models, we gave students the flexibility to express and explore their own understanding.

In particular, we have suggested some principles for the design of learner-centered software, as they relate to software tasks, tools, and interfaces:

• *Tasks* In order to support learning, the software must be scaffolded to the level of the student. As the data showed, by providing a constrained set of domain-specific object primitives, and a qualitative means of expression, students achieved early and motivating success in constructing their models.

• *Tools* Students are constantly changing and growing, and the software must be flexible enough adapt to their level of expertise. By providing multiple means of expression, qualitative and quantitative, Model-It supports students as they learn and develop modeling experience.

• *Interfaces* To engage the learner, we provide multiple visual representations of the concepts being explored. With personally meaningful photo-realistic images, text graphs, and dynamic meters, students were able to visualize and interpret their models.

In summary, we wish to re-emphasize the distinction between the user and the learner: if the goal is addressing the needs of the user, the focus should be on ease of use. However, if the goal is addressing the needs of the learner, the focus must be on providing support and motivation for the growth of understanding and expertise.

# Acknowledgments

The authors would like to extend our great appreciation to the other members of the ScienceWare project, especially Jiannchuan Tony Hu and Robert Royce who contrbuted to the design and implementation of Model-It, Liza Finkel, David Schmidt and Jeff Spitulnik who participated in the user testing, and Mark Guzdial who participated in conversations about Learner-Centered Design. The authors would also like to thank the teachers and students at Community High School in Ann Arbor, for their feedback and support.

This research has been supported by the National Science Foundation (RED 9353481) and the University of Michigan.

# REFERENCES

1. Blumenfeld, P., Soloway, E., Marx, R. W., Krajcik, J. S., Guzdial, M., & Palincsar, A. (1991) Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning, *Educational Psychologist*, Vol. 26.(3 & 4), 369-398

2. Guzdial, M. (1993) Emile: Software-Realized Scaffolding for Science Learners Programming in Mixed Media. Unpublished Ph.D. dissertation, University of Michigan.

3. Harel, I., & Papert, S. (1990) Software design as a learning environment. *Interactive Learning Environments* , 1(1), 1-32.

4. Kreutzer, W. (1986) *Systems Simulation: Programming Styles and Languages,* Addison-Wesley, Wokingham, England.

5. Norman, D., Draper, S. (1986) *User Centered System Design,* L. Erlbaum & Assoc, Hillsdale, NJ.

6. Palincsar, A. S. (1986) The role of dialogue in providing scaffolded instruction. *Educational Psychologist* , 21(1-2), 73-98.

7. Papert, S. (1991) Situating constructionism. In I. Harel & S. Papert (Eds.), *Constructionism* . Norwood, NJ: Ablex Publishing Company. pp. 1-11.

8. Scardamalia, M., & Bereiter, C. (1991) Higher levels of agency for children in knowledge building: A challenge for the design of new knowledge media. *Journal of the Learning Sciences* , 1(1), 37-68.

9. Soloway, E., Guzdial, M., & Hay, K. E. (1994) Learner-Centered Design: The Challenge for HCI in the 21st Century, *Interactions* , Vol. 1, No. 2, April, 36-48

10. Tinker, R. F. (1990) Teaching Theory Building: Modeling: Instructional Materials and Software for Theory Building, NSF Final Report, TERC.

11. Wood, D., Bruner, J. S., & Ross, G. (1975). The role of tutoring in problem-solving. *Journal of Child Psychology and Psychiatry* , 17, 89-100.